REMARKS

Applicants respectfully request reconsideration and allowance of the present application. By this Amendment, Applicants amend claims 1, 9, 11, 13, 15 and 16. Claims 1-52 will remain pending in the application upon entry of this Amendment.

Rejection of Claims Under 35 U.S.C. § 102(e)

In the Office Action, claims 1-3 under 35 USC 102(e) stand rejected as allegedly being taught by U.S. Patent No. 6,085,315 to Fleck et al. ("Fleck"). For reasons set forth more fully below, this rejection is respectfully traversed.

Fleck Does Not Store A Kernel Set of Loop Body Instructions As Required By Amended Independent Claim 1

Amended independent claim 1 requires, *inter alia*, a buffer in a dispatch stage that is adapted to store a plurality of instructions **wherein the plurality of the instructions comprise a kernel set of instructions from a loop body.**[1]

The Office Action alleges that Fleck's "loop cache 3" corresponds to the claimed buffer. However, Fleck's "loop cache 3" only stores a loop instruction, and does not store any instructions from a loop body, much less a kernel set of instructions from a loop body as required by amended independent claim 1.

This fact is clear throughout Fleck's descriptions, particularly in connection with FIGs. 2 and 3. Specifically, as set forth in col. 3, lines 31-41, Fleck teaches that:

> FIG. 2 shows the special register which is part of the loop cache buffer 3. A first portion 3a of the register content represents the previous instruction address. A second portion 3b represents the target instruction address. A third portion 3c represents the loop instruction size and a fourth portion 3d indicates that the entries are valid. Preferably, the size of the address portions 3a and 3b complies with the respective address range or program counter. The loop instruction size portion 3c indicates the word size of the loop instruction. A loop instruction may be, for example, a 16 bit or a 32 bit instruction.

[1] This limitation was already present in certain dependent claims and independent claims that the Examiner has already reviewed. Accordingly, no new issues are raised requiring further examination, and Applicants respectfully request a new Office Action should a Notice of Allowance not result from this response.

Moreover, at col. 31 lines 48-54, Fleck teaches that:

> FIG. 3 shows parts of the program counter update and control unit 2, parts of loop target buffer 3, and parts of loop execution unit 4 in more detail. Same elements have the same numerals. The loop target buffer 3 contains the register 3a, 3b, 3c, 3d shown in FIG. 2. In addition, a register 3e is provided which stores the loop instruction detected during the first iteration.

Still further, at col. 4, lines 9-16, Fleck teaches that:

> After processing the loop for the first time, the loop cache buffer 3 is set. In other words, the target address is stored in register portion 3b, the previous instruction address is stored in register portion 3a, and the loop instruction size is stored in register portion 3c. Also, register portion 3c which can be a flag, is set to indicate that the entries are valid. In addition, the loop instruction itself is stored in a register 3e which outputs its content to the multiplexer 5.

To assist the Examiner's review, FIGs. 2 and 3 from Fleck are reproduced below

FIG. 2

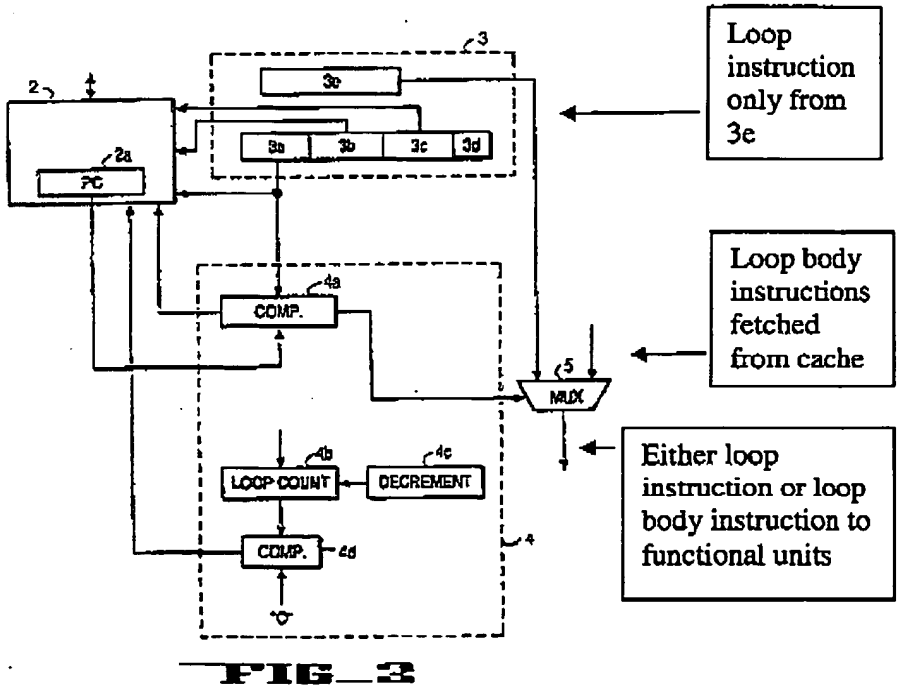

FIG. 3

Fleck is aimed merely at providing special hardware and a dedicated "loop pipeline" so as to facilitate no-overhead loops. As set forth in Fleck's summary at col. 1, lines 40-49:

> The loop pipeline can be a pipeline with a functionality only directed to the special loop instruction. It therefore does not necessarily need all the operating units of a regular pipeline. The loop pipeline handles the zero overhead loops, such that during the repetition of a loop, no branch or jump instruction and no condition testing are executed in the main pipeline. The loop pipeline is used to fold out the loop instruction from the execution flow allowing a loop instruction to be performed in parallel, for example, with an integer and load/store operation.

Clearly, therefore, Fleck's loop cache 3 is intended only to cache the loop instruction so that it can be provided to the special loop pipeline at appropriate times via multiplexer 5. Fleck still requires that loop body instructions be fetched from an instruction cache via multiplexer 5 and provided in separate pipelines normally.

In stark contrast, the claimed invention provides that a kernel set of **loop body** instructions **are buffered in the dispatch stage of the processor**.

For at least these reasons, amended independent claim 1 patentably defines over Fleck and the § 102 rejection of claim 1, together with claims 2 and 3 that depend therefrom, should be withdrawn.

Fleck's "Loop Cache 3" Is Not In a Dispatch Stage As Required By Amended Independent Claim 1

Amended independent claim 1 requires, *inter alia*, "a buffer **in the dispatch stage** coupled to the functional unit adapted to store a plurality of the instructions before issue to the functional unit." The Office Action indicates that this claimed subject matter is found Fleck. Applicants respectfully disagree.

First, Fleck's loop cache 3 stores a single loop instruction to be fed to the "instruction providing unit." In particular, in Figure 1 of Fleck, the loop cache feeds into the Instruction Demux (Box 7) via the Mux (Box 5). Fleck specifically refers to instruction demux 7 as an "instruction fetch unit" (col. 1, lines 31-32). Moreover, the loop cache is included before demux 7 in "instruction fetch module 1", which Fleck describes at col. 3, lines 1-8 as the "fetch stage."

Accordingly, one skilled in the art would not consider Fleck's loop cache to be **in the dispatch stage** of the processor. Rather, Fleck's loop cache 3 is clearly and explicitly in a **fetch stage**.

For at least these additional reasons, the § 102 rejection of claim 1, along with claims 2 and 3 that depend therefrom, should be withdrawn.

Claim 3 Further Patentably Defines Over Fleck

Claim 3 depends from claim 1, which has been shown above to patentably define over Fleck. Accordingly, claim 3 is patentable for at least the reasons claim 1 is patentable.

Claim 3 further requires "control logic coupled to the buffer adapted to cause a **certain** one of the stored plurality of instructions to be issued to the functional unit in accordance with a loop iteration stage." The Office Action indicates that this subject matter is met by Fleck (column 2, lines 45-67; column 4, lines 9-56; Figure 1; and Figure 3). Applicants respectfully disagree.

First, similarly as set forth above, whereas the claim requires that the control logic identifies **certain** ones of the **stored** instructions to be issued to a functional unit, Fleck's alleged buffer (i.e. "loop cache 3") only stores a single loop instruction for provision to instruction demux 7 via multiplexer 5. Therefore, there is no structure that permits certain ones of a plurality of instructions to be issued from cache 3 to a functional unit, much less based on a loop iteration stage as required by claim 3.

For at least this additional reason, the § 102 rejection of claim 3 should be withdrawn.

Rejection of Claims Under 35 U.S.C. § 103(a) By Fleck and Subramanian

Claims 4-18, 26-33, 35-41 and 43-49 stand rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Fleck in view of U.S. Patent No. 5,867,711 to Subramanian et al. ("Subramanian"). For reasons set forth more fully below, this rejection is respectfully traversed.

Claims 4-18 Patentably Define Over Fleck and Subramanian By Dependence On Claim 1

Claims 4-18 depend from amended independent claim 1, and certain other claims further depend directly or indirectly from claim 3, both of which have been shown above to patentably

define over Fleck. The alleged combination with Subramanian would not have suggested the subject matter of claims 1 and 3 that is not missing from Fleck, and the Office Action correctly fails to allege as such.

More particularly, Fleck merely teaches a loop cache 3 that allows a loop instruction to be provided in parallel with loop body instructions during execution of a loop. Subramanian merely teaches a compiler that is aimed at improved scheduling of loop execution. Accordingly, the alleged combination of Fleck and Subramanian would result in Fleck's processor that only buffers a single loop instruction, plus an improved compiler. It would not result in or suggest a buffer **in a dispatch stage that stores a kernel set of instructions from a loop body** as required by amended independent claim 1.

Accordingly, claims 4–18 are patentable for at least the reasons claims 1 and 3 are patentable.

Claim 4 Further Patentably Defines Over Fleck and Subramanian

Claim 4 depends from patentable claims 1 and 3 and further requires that the control logic causes the **certain** stored instructions to be issued from the buffer to the functional unit "in accordance with **a cycle within the loop iteration stage**." The invention of claim 4 thus requires a measure of the number of cycles (i.e. "loop cycles") in the loop body, whereas neither Fleck nor Subramanian teaches or suggests such a measure to be tracked by the control logic in a processor.[2]

First, it is important to note that Subramanian teaches compiler software methods for creating a modulo schedule of a loop iteration, which is an unrelated art from processor design. Thus, Subramanian does not teach, and in fact **teaches away** from, providing any control logic **in a processor** adapted to cause the certain ones of the instructions to be issued **in accordance with a cycle within the loop iteration stage** (see the present specification at, for example, page 3, line 19 to page 4 line 3, page 6, line 22 to page 7 line 4, and page 7, lines 14-20).

[2] In the Final Office Action, the Examiner suggests that the claim "merely states that a loop instruction is executed in accordance with which loop iteration the loop is in." Applicants disagree. The claim clearly requires control logic **in the processor** that is adapted to cause first and second different instructions stored in **a buffer in the processor dispatch stage** to be issued to the functional unit.

Subramanian aims to derive the loop stages and cycle schedule from a dependence graph. As taught by Subramanian, the schedule is then represented in an instruction sequence that consists of a Prolog, Kernel and Epilog (PKE form). In Subramanian's teaching, the prolog and epilog code sequences are explicitly generated by the compiler software and provided to the processor in that form. The target processor hardware itself is not aware of the concept of loop iteration stages and loop cycles, and so one skilled in the art would not incorporate structure in Fleck's processor as a result of Subramanian's teachings, even if the teachings were from related arts.

In stark contrast, the present invention provides a hardware apparatus that can allow the compiler software to efficiently communicate the number of loop iteration stages and cycles of its derived schedule to hardware. In accordance with the present invention, a compiler is not required to explicitly generate entire instruction sequences PKE form. Rather, the invention allows the compiler to compactly represent the schedule in the instruction stream. The control unit then dynamically derives the loop iteration stage and cycles information needed to cause the appropriate instructions in each buffer to be issued to the associated function unit.

To summarize, Subramanian and its associated prior art teach how to reach a schedule from a dependence graph in compiler software, whereas the present invention teaches how to enable compact representation of the derived schedule and efficient hardware buffering and execution.

As set forth above, claim 4 requires control logic **in the processor** that maintains a **loop cycles counter** as a pointer that iterates through the buffer. The claimed control logic uses this pointer to issue successive instructions out of the buffers. In contrast, Fleck relies on the program counter values. Accordingly, even if one could apply the teachings of Subramanian to Fleck, the alleged combination would still not have suggested the subject matter of claim 4 because the alleged combination would still have only allowed entire loop instruction sequences (downloaded from Subramanian's compiler solution and stored in Fleck's cache subsystem) to be issued by Fleck's processor in accordance with program counter values. The invention of claim 4, however, requires instructions to be issued using control logic in the processor which causes them to be issued in accordance with a cycle within the loop iteration stage.

For at least the foregoing reasons, claim 4, together with claim 7 that depends therefrom, patentably define over Fleck and Subramanian and the § 103 rejection thereof should be withdrawn.

Amended Claims 15 and 16 Further Patentably Defines Over Fleck and Subramanian

Amended claims 15 and 16 ultimately depend from patentable claim 1 and further require that the control logic is operative so that the fetch unit can be shut down during execution of "**a number of iterations** of the **loop body corresponding to the** stored plurality of instructions."

In contrast, Fleck uses Program Counter 2, **which is part of the fetch module 1**, to control fetches from an instruction cache to access loop body instructions during execution of a loop. This is clearly illustrated by FIG. 5 of Fleck, and the corresponding description of the operation of Fleck's processor during execution of iterations of the loop in columns 5 and 6. As can be clearly seen, Fleck's processor requires loop body instructions A1, A2, L1, L2 to be fetched during execution of loop iterations.

INTEGER PIPE	FETCH	A1	A2	A1	A2	...
	DECODE	A2	A1	A2	A1	...
	EXECUTE	A1	A2	A1	A2	...
	WRITE BACK	-	A1	A2	A1	...
LOAD / STORE PIPE	FETCH	L1	L2	L1	L2	...
	DECODE	L2	L1	L2	L1	...
	EXECUTE	L1	L2	L1	L2	...
	WRITEBACK	-	L1	L2	L1	...
LOOP PIPE	DECODE	-	B	-	B	...
	WRITEBACK	-	-	B	-	...
		C6	C7	C8	C9	C10

FIG. 5

Fleck merely suggests that the single loop instruction itself does not need to be fetched. However, Fleck does not suggest or teach that the entire fetch unit can be shut down for all iterations of a loop as required by claims 15 and 16.

For at least the foregoing reasons, claims 15 and 16 further patentably define over Fleck and Subramanian and the § 103 rejection thereof should be withdrawn.

Independent Claim 26 Patentably Defines Over Fleck and Subramanian

Independent claim 26 requires that:

- The buffer is "in the dispatch stage";
- The buffer is "associated with a functional unit";
- The buffer has "a first portion for storing a kernel set of instructions"; and
- The buffer has "a second portion for storing a plurality of modulo schedule stage identifiers"

As set forth in more detail above in connection with claim 1, Fleck's alleged buffer (loop cache 3) is not "in the dispatch stage," but before it (i.e. instruction demux 7). More particularly, Fleck's loop cache 3 only stores a single loop instruction, and does not contain the first and second portions required by claim 26, particularly including a portion "storing a kernel set of instructions." The Office Action admits this, but relies on Subramanian for this subject matter.

First, as set forth more fully above in connection with claim 4, Subramanian teaches compiler software methods for creating a modulo schedule of a loop iteration, and thus **teaches away** from, providing any modulo scheduling logic **in a processor**. Rather, in Subramanian's teaching, the prolog, kernel and epilog code sequences are explicitly generated by the compiler software and provided to the processor in that form. The target processor hardware itself is not aware of the concept of loop iteration stages and loop cycles, much less a kernel set of loop instructions.

In stark contrast, the present invention provides a hardware apparatus that can allow the compiler software to efficiently communicate the number of loop iteration stages and cycles of its derived schedule to hardware. In accordance with the present invention, a compiler is not required to explicitly generate entire instruction sequences PKE form. Rather, the invention allows the compiler to compactly represent the schedule in the instruction stream.

As set forth above, claim 26 requires a buffer in the processor having separate portions for storing:

- A kernel set of loop instructions; and
- A plurality of modulo schedule stage identifiers respectively associated with the kernel set of loop instructions.

Subramanian teaches away from providing such a buffer in the processor, and in any event does not teach a buffer having the required first and second portions. Accordingly, even if one could apply the teachings of Subramanian to Fleck, the alleged combination would still not have suggested the subject matter of claim 26.

For at least the foregoing reasons, claim 26, together with claims 27-31 that depend therefrom, patentably define over Fleck and Subramanian and the § 103 rejection thereof should be withdrawn.

Amended Independent Claim 32 Patentably Defines Over Fleck and Subramanian

Amended independent claim 32 requires that:

- The buffers are in "the dispatch stage";
- The buffers are "respectively associated with" the plurality of functional units;
- The buffers in the processor "store the kernel set of loop instructions"; and
- Control logic in the processor is coupled to the buffers "for causing the stored kernel set of instructions to be selectively issued to the functional units".

As set forth in more detail above in connection with claim 1, Fleck's alleged buffer (loop cache 3) is not "in the dispatch stage," but before it (i.e. instruction demux 7), and Fleck's alleged buffer only stores a single loop instruction, as opposed to the claimed "kernel set of loop instructions."

Moreover, Fleck's loop cache 3 is not a plurality of buffers "respectively associated with" the plurality of functional units, but rather this one cache serves a single "loop pipeline."

In still further contrast, as set forth more fully above in connection with claim 4, Subramanian teaches **compiler software** methods for creating a modulo schedule of a loop iteration, and thus **teaches away** from, providing any modulo scheduling control logic **in a processor**. Rather, in Subramanian's teaching, the prolog, kernel and epilog code sequences are explicitly generated by the compiler software and provided to the processor in that form. The target processor hardware itself is not aware of the concept of loop iteration stages and loop cycles, much less a kernel set of loop instructions.

In stark contrast, the present invention provides a hardware apparatus that can allow the compiler software to efficiently communicate the number of loop iteration stages and cycles of its derived schedule to hardware. In accordance with the present invention, a compiler is not required to explicitly generate entire instruction sequences PKE form. Rather, the invention allows the compiler to compactly represent the schedule in the instruction stream.

As set forth above, claim 32 requires a buffer and associated control logic **in the processor** wherein:

- The buffers store a kernel set of loop instructions; and
- The control logic causes the stored kernel set of instructions to be selectively issued to the functional units based on the stored kernel set of loop instructions.

Subramanian teaches away from providing such buffers and control logic **in the processor**, and in any event does not teach the buffers and control logic required by claim 32. Accordingly, even if one could apply the teachings of Subramanian to Fleck, the alleged combination would still not have suggested the subject matter of claim 32.

For at least the foregoing reasons, claim 32, together with claims 33 and 34 that depend therefrom, patentably define over Fleck and Subramanian and the § 103 rejection thereof should be withdrawn.

Independent Claims 35 and 43 Patentably Define Over Fleck and Subramanian

Independent claims 35 and 43 require that:

- The kernel set of loop instructions are stored in "a dispatch stage" of the processor;
- Loop parameters are stored "in control logic of the processor"; and
- The stored kernel set of loop instructions are caused "to be selectively issued to functional units of the processor" in accordance with the stored loop parameters.

As set forth in more detail above in connection with claim 1, Fleck's alleged buffer (loop cache 3) is not "in the dispatch stage," but before it (i.e. instruction demux 7), and it only stores a single loop instruction, and not a kernel set of loop instructions.

In still further contrast, as set forth more fully above in connection with claim 4, Subramanian teaches **compiler software** methods for creating a modulo schedule of a loop

iteration, and thus teaches away from, providing any modulo scheduling control logic in a processor. Rather, in Subramanian's teaching, the prolog, kernel and epilog code sequences are explicitly generated by the compiler software and provided to the processor in that form. The target processor hardware itself is not aware of the concept of loop iteration stages and loop cycles, much less a kernel set of loop instructions.

In stark contrast, the present invention provides a hardware apparatus that can allow the compiler software to efficiently communicate the number of loop iteration stages and cycles of its derived schedule to hardware. In accordance with the present invention, a compiler is not required to explicitly generate entire instruction sequences PKE form. Rather, the invention allows the compiler to compactly represent the schedule in the instruction stream.

As set forth above, claim 35 and 43 require storing loop instructions and associated control logic in the processor wherein:

- The dispatch stage includes means for storing a kernel set of loop instructions; and
- Control logic stores loop parameters for causing the stored kernel set of instructions to be selectively issued to the functional units based on the stored kernel set of loop instructions.

Subramanian teaches away from providing such storage and control logic in the processor, and in any event does not teach the storage and control logic required by claims 35 and 43. Accordingly, even if one could apply the teachings of Subramanian to Fleck, the alleged combination would still not have suggested the subject matter of claims 35 and 43.

For at least the foregoing reasons, claims 35 and 43, together with claims 36-42 and 44-50 that respectively depend therefrom, patentably define over Fleck and Subramanian and the § 103 rejection thereof should be withdrawn.

Rejection of Claims Under 35 U.S.C. § 103(a) By Fleck, Subramanian and Valluri

The Examiner has rejected claims 19-21 under 35 USC 103(a) as allegedly being unpatentable over Fleck in view of Subramanian as applied to claims 9, 11 and 13, above, and further in view of Valluri and Govindarajan's "Modulo-Variable Expansion Sensitive Scheduling" published in *High Performance Computing*, 1998.

Claims 19-21 depend from claims 9, 11 and 13, which have been shown above to patentably define over Fleck and Subramanian, at least because Fleck and Subramanian do not suggest a buffer in the dispatch stage as required by independent claim 1. The further alleged combination of Valluri with Fleck and Subramanian would not cure this deficiency.

Moreover, Valluri is directed to compiler-based scheduling of code, which further teaches away from the invention, which is based in hardware. Accordingly, one skilled in the art would not be led to the hardware-based invention of claims 19-21, even if, *arguendo*, Valluri could be combined with Fleck and Subramanian.

For at least these reasons, the rejections of claims 19-21 should be withdrawn.

Rejection of Claims Under 35 U.S.C. § 103(a) By Fleck, Subramanian and Morrison

Additionally, claims 22-25, 34, 42 and 50 were rejected under 35 USC 103(a) as allegedly being unpatentable over Fleck in view of Subramanian as applied to claims 3, 8, 11 and 13, above, and further in view of U.S. Patent No. 6,421,744 to Morrison et al. ("Morrison").

Claims 22-25 depend ultimately from independent claim 1, claim 34 depends from independent claim 32, claim 42 depends from independent claim 35, and claim 50 depends from independent claim 43. These independent claims have been shown above to patentably define over Fleck and Subramanian. The further alleged combination of these references with Morrison would not overcome the shortcomings of Fleck and Subramanian as discussed above. Accordingly, claims 22-25, 34, 42 and 50 are patentable at least for the reasons claims 1, 32, 35 and 50 are patentable.

Moreover, Morrison does not teach the type of loop iteration required in the rejected claims. For example, claim 34 requires the control logic to be "operative to allow interrupts to be handled at the end of a current one of the number of loop iterations, and to complete the number of loop iterations after the interrupt is handled." In contrast Morrison merely discloses taking interrupts at the end of loop iterations. In modulo scheduled code according to the present invention, there is no natural and clean end of an iteration; all the iterations are overlapped. Accordingly, Morrison's interrupts would not meet the limitations explicitly required by the claims.

For the foregoing reasons, claims 22-25, 34, 42 and 50 patentably define over the cited prior art and the § 103 rejection of the claims should be withdrawn.

Conclusion

All objections and rejections having been addressed, and in view of the foregoing, the claims are believed to be in form for allowance, and such action is hereby solicited. If any points remain in issue which the Examiner feels may be best resolved through a personal or telephone interview, he or she is kindly requested to contact the undersigned at the telephone number listed below.

Respectfully submitted,
PILLSBURY WINTHROP LLP

Date: Feb. 22, 2005

Mark J. Danielson 40,580
Reg. No.
(650) 233-4777
Please reply to customer no. 27,498